Claims

WHAT IS CLAIMED IS:

1. A method comprising:

receiving into an execution environment input component code and a runtime security policy; and

generating a call graph of call paths through the input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with the runtime security policy.

2. The method of claim 1 wherein a possible execution path through the input component code that is compliant with the runtime security policy is represented by an individual call path.

3. The method of claim 1 wherein at least one node in the call graph includes a symbolic permission set and a known method implementation.

4. The method of claim 1 wherein at least one node in the call graph includes a symbolic permission set and a token representing an unknown method implementation.

5. The method of claim 1 wherein the generating operation comprises:

initializing a symbolic value that represents data values that may be obtained by the arbitrary code at runtime.

6. The method of claim 1 wherein the generating operation comprises:

updating a symbolic value that represents data values that may be obtained by the arbitrary code at runtime based on detection of an additional data value that may be passed as a parameter to the arbitrary code at runtime.

7. The method of claim 1 wherein the generating operation comprises:

updating a symbolic value that represents data values that may be obtained by the arbitrary code at runtime based on detection of a new dataflow to the arbitrary code.

8. The method of claim 1 wherein the generating operation comprises:

generating a class hierarchy that contains classes of the input component code and symbolic classes that represent classes of arbitrary code.

9. The method of claim 1 wherein the generating operation comprises:

identifying one or more methods of the input code component that can be called by the arbitrary code.

10. The method of claim 1 wherein the generating operation comprises:

identifying one or more methods of the input component code that can be called by the arbitrary code; and

identifying one or more other methods of the input component code that can be called by the identified one or more methods of the input component code.

11. The method of claim 1 wherein the generating operation comprises:

identifying one or more methods of the input component code that can be called by the arbitrary code; and identifying at least one method of the arbitrary code that can be called by a virtual call of the identified one or more methods of the input component code.

12. The method of claim 1 wherein any method reachable by execution in accordance with the runtime security policy is represented by one of more nodes in the call graph.

13. The method of claim 1 wherein the generating operation comprises:

generating at least one constraint associated with one or more instructions in the input component code.

14. The method of claim 1 wherein the generating operation comprises:

generating at least one constraint associated with a parameter of a method call in the input component code.

15. The method of claim 1 wherein the generating operation comprises:

generating at least one constraint associated with a returned result of a method call in the input component code.

16. The method of claim 1 further comprising:

analyzing the call graph to identify a call path that presents a security vulnerability in the input component code.

17. The method of claim 1 further comprising:

analyzing the call graph to identify a call path that presents a security vulnerability in the input component code and a call path that presents no security vulnerability in the input component code.

18. The method of claim 1 further comprising:

analyzing the call graph to generate a security report that identifies a security vulnerability in the input component code.

19. The method of claim 1 further comprising:

analyzing the call graph to identify a call path that satisfies a query.

20. The method of claim 1 further comprising:

analyzing the call graph to identify a security-vulnerable usage of a permission demand.

21. The method of claim 1 further comprising:

analyzing the call graph to identify a security-vulnerable usage of a permission assertion.

22. The method of claim 1 further comprising:

analyzing the call graph to identify a lack of uniform usage of security checks.

23. The method of claim 1 further comprising:

analyzing the call graph to identify an equivalence between use of a permission link-demand and a permission demand.

24. The method of claim 1 wherein the generating operation comprises:

generating a class hierarchy that contains classes of the input component code and symbolic classes that represent classes of the arbitrary code;

generating at least one constraint associated with a virtual call in the input component code; and evaluating the at least one constraint by a symbolic computation on potential target classes for the virtual call in the generated class hierarchy.

25. The method of claim 1 wherein the generating operation comprises:

generating at least one constraint associated with either a security demand or a security assert in the input component code;

evaluating the at least one constraint by a symbolic computation on dynamic permissions of the input component code and on a parameter permission of the security demand or the security assert; and

conditionally generating at least one additional constraint associated with one or more instructions located in the input component code after the security demand or assert, responsive to the evaluating operation.

26. The method of claim 1, further comprising

analyzing the call graph to classify, based on permissions, pieces of code performing sensitive actions in the input component code.

27. The method of claim 1, further comprising

analyzing the call graph and another call graph obtained for a different version of input component code to generate a security report that identifies a security vulnerability in the different version of the input component code.

28. The method of claim 1, further comprising

analyzing the call graph and another call graph obtained for a different version of input component code to identify a call path that presents a security vulnerability in the different version of the input component code.

29. A computer program product encoding a computer program for executing on a computer system a computer process, the computer process comprising:

receiving into an execution environment input component code and a runtime security policy; and

generating a call graph of call paths through the input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with the runtime security policy.

30. The computer program product of claim 29 wherein a possible execution path through the input component code that is compliant with the runtime security policy is represented by an individual call path.

31. The computer program product of claim 29 wherein at least one node in the call graph includes a symbolic permission set and a known method implementation.

32. The computer program product of claim 29 wherein at least one node in the call graph includes a symbolic permission set and a token representing an unknown method implementation.

33. The computer program product of claim 29 wherein the generating operation comprises:

initializing a symbolic value that represents data values that may be obtained by the arbitrary code at runtime.

34. The computer program product of claim 29 wherein the generating operation comprises:

updating a symbolic value that represents data values that may be obtained by the arbitrary code at runtime based on detection of an additional data value that may be passed as a parameter to the arbitrary code at runtime.

35. The computer program product of claim 29 wherein the generating operation comprises:

updating a symbolic value that represents data values that may be obtained by the arbitrary code at runtime based on detection of a new dataflow to the arbitrary code.

36. The computer program product of claim 29 wherein the generating operation comprises:

generating a class hierarchy that contains classes of the input component code and symbolic classes that represent classes of arbitrary code.

37. The computer program product of claim 29 wherein the generating operation comprises:

identifying one or more methods of the input component code that can be called by the arbitrary code.

38. The computer program product of claim 29 wherein the generating operation comprises:

identifying one or more methods of the input component code that can be called by the arbitrary code; and identifying one or more other methods of the input component code that can be called by the identified one or more methods of the input component code.

39. The computer program product of claim 29 wherein the generating operation comprises:

identifying one or more methods of the input component code that can be called by the arbitrary code; and

identifying at least one method of the arbitrary code that can be called by a virtual call of the identified one or more methods of the input component code.

40. The computer program product of claim 29 wherein any method reachable by execution in accordance with the runtime security policy is represented by one of more nodes in the call graph.

41. The computer program product of claim 29 wherein the generating operation comprises:

generating at least one constraint associated with one or more instructions in the input component code.

42. The computer program product of claim 29 wherein the generating operation comprises:

generating at least one constraint associated with a parameter of a method call in the input component code.

43. The computer program product of claim 29 wherein the generating operation comprises:

generating at least one constraint associated with a returned result of a method call in the input component code.

44. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to identify a call path that presents a security vulnerability in the input component code.

45. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to identify a call path that presents a security vulnerability in the input component code and a call path that presents no security vulnerability in the input component code.

46. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to generate a security report that identifies a security vulnerability in the input component code.

47. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to identify a call path that satisfies a query.

48. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to identify a security-vulnerable usage of a permission demand.

49. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to identify a security-vulnerable usage of a permission assertion.

50. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to identify a lack of uniform usage of security checks.

51. The computer program product of claim 29 wherein the computer process further comprises:

analyzing the call graph to identify an equivalence between use of a permission link-demand and a permission demand. [Inventors: Can you define a link-demand? Is this a Demand at link time? If so, I will use that definition instead of the specific term link-demand.

52. The computer program product of claim 29 wherein the generating operation comprises:

generating a class hierarchy that contains classes of the input component code and symbolic classes that represent classes of the arbitrary code;

generating at least one constraint associated with a virtual call in the input component code; and

evaluating the at least one constraint by a symbolic computation on potential target classes for the virtual call in the generated class hierarchy.

53. The computer program product of claim 29 wherein the generating operation comprises:

generating at least one constraint associated with either a security demand or a security assert in the input component code;

evaluating the at least one constraint by a symbolic computation on dynamic permissions of the input component code and on a parameter permission of the security demand or the security assert; and

conditionally generating at least one additional constraint associated with one or more instructions located in the input component code after the security demand or assert, responsive to the evaluating operation.

54. The computer program product of claim 29 further comprising

analyzing the call graph to classify, based on permissions, pieces of code performing sensitive actions in the input component code.

55. The computer program product of claim 29 further comprising

analyzing the call graph and another call graph obtained for a different version of input component code to generate a security report that identifies a security vulnerability in the different version of the input component code.

56. The computer program product of claim 29 further comprising

analyzing the call graph and another call graph obtained for a different version of input component code to identify a call path that presents a security vulnerability in the different version of the input component code.

57. A system comprising:

a call graph generator receiving into an execution environment input component code and a runtime security policy, and generating a call graph of call paths through the input component code simulated in combination with at least one symbolic component that represents additional arbitrary code that complies with the runtime security policy.

58. The system of claim 57 wherein a possible execution path through the input component code that is compliant with the runtime security policy is represented by an individual call path.

59. The system of claim 57 further comprising:

a call graph analyzer that analyzes the call graph to identify a security-vulnerable usage of a permission demand.

60. The system of claim 57 further comprising:

a call graph analyzer that analyzes the call graph to identify a security-vulnerable usage of a permission assertion.

61. The system of claim 57 further comprising:

a call graph analyzer that analyzes the call graph to identify a lack of uniform usage of security checks.

62. The system of claim 57 further comprising:

a call graph analyzer that analyzes the call graph to identify an equivalence between use of a permission link-demand and a permission demand.

63. A method comprising:

analyzing relative to at least one query a call graph of call paths through input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with a runtime security policy; and

identifying a subset of the call paths in the call graph that satisfy the query.

64. The method of claim 63 wherein the analyzing operation comprises:

analyzing the call graph to identify a security-vulnerable usage of a permission demand.

65. The method of claim 63 wherein the analyzing operation comprises:

analyzing the call graph to identify a security-vulnerable usage of a permission assertion.

66. The method of claim 63 wherein the analyzing operation comprises:

analyzing the call graph to identify a lack of uniform usage of security checks.

67. The method of claim 63 wherein the analyzing operation comprises:

analyzing the call graph to identify an equivalence between use of a permission link-demand and a permission demand.

68. A computer program product encoding a computer program for executing on a computer system a computer process, the computer process comprising:

analyzing relative to at least one query a call graph of call paths through input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with a runtime security policy; and

identifying a subset of the call paths in the call graph that satisfy the query.

69. The computer program product of claim 68 wherein the compute process further comprises:

analyzing the call graph to identify a security-vulnerable usage of a permission demand.

70. The computer program product of claim 68 wherein the compute process further comprises:

analyzing the call graph to identify a security-vulnerable usage of a permission assertion.

71. The computer program product of claim 68 wherein the compute process further comprises:

analyzing the call graph to identify a lack of uniform usage of security checks.

72. The computer program product of claim 68 wherein the compute process further comprises:

analyzing the call graph to identify an equivalence between use of a permission link-demand and a permission demand.

73. A system comprising:

a call graph analyzer analyzing relative to at least one query a call graph of call paths through input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with a runtime security policy, and identifying a subset of the call paths in the call graph that satisfy the query.

74. The system of claim 73 wherein the call graph analyzer analyzes the call graph to identify a security-vulnerable usage of a permission demand.

75. The system of claim 73 wherein the call graph analyzer analyzes the call graph to identify a security-vulnerable usage of a permission assertion.

76. The system of claim 73 wherein the call graph analyzer analyzes the call graph to identify a lack of uniform usage of security checks.

77. The system of claim 73 wherein the call graph analyzer analyzes the call graph to identify an equivalence between use of a permission link-demand and a permission demand.